Efficient 3D Movement-Based Kernel Density Estimator and Application to Wildlife Ecology

Jeff A. Tracey
Western Ecological Research Center
U.S. Geological Survey
San Diego, CA 92101
jatracey@usgs.gov

James K. Sheppard
San Diego Zoo
Institute for Conservation Research
Escondido, CA 92027
jsheppard@sandiegozoo.org

Glenn K. Lockwood
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA 92093
glock@sdsc.edu

Amit Chourasia
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA 92093
amit@sdsc.edu

Mahidhar Tatineni
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA 92093
Mahidhar@sdsc.edu

Robert N. Fisher
Western Ecological Research Center
U.S. Geological Survey
San Diego, CA 92101
rfisher@usgs.gov

Robert S. Sinkovits
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA 92093
sinkovit@sdsc.edu

ABSTRACT

We describe an efficient implementation of a 3D movement-based kernel density estimator for determining animal space use from discrete GPS measurements. This new method provides more accurate results, particularly for species that make large excursions in the vertical dimension. The downside of this approach is that it is much more computationally expensive than simpler, lower-dimensional models. Through a combination of code restructuring, parallelization and performance optimization, we were able to reduce the time to solution by up to a factor of 1000x, thereby greatly improving the applicability of the method.

Categories and Subject Descriptors

D.1.3 Parallel programming, H.3.4 Performance Evaluation, J.2 Mathematics and statistics, J.3 Biology and genetics

General Terms

Performance, Algorithms

Keywords

Parallel computing, performance optimization, biotelemetry, wildlife ecology, visualization

XSEDE '14, July 13 - 18 2014, Atlanta, GA, USA

1. INTRODUCTION

The increasing sophistication and miniaturization of digital biotelemetry tracking devices (biologgers) has enabled researchers to collect large, highly accurate and long-term datasets on the movements of free-ranging animals that would normally be prohibitively difficult to observe directly in the wild [1-3]. Global Positioning System (GPS) biologgers can now record an animal's geographic coordinates (e.g. latitude and longitude) at accuracies to within 2m for deployments lasting more than a year, or even longer if powered by a solar panel. GPS biologgers have also dramatically reduced in size and can now be safely attached to small animals. For example, the California condors reintroduced to their former habitat in Mexico by San Diego Zoo Global (SDZG) have a <50g solar-powered GPS biologger attached to their wings. These condor tags provide hourly location fixes from each bird at a resolution of just a few meters that can be downloaded directly from the Internet [4].

Advances in biotelemetry technologies have contributed to major advances in our understanding of key concepts of animal ecology, including resource use, home range, dispersal, and population dynamics [5]. Biotelemetry technologies are also becoming powerful tools for informing strategies for conserving endangered species and habitats. For example, GPS biologgers can provide accurate information on the movements of a tracked animal that can then be matched to the environmental attributes that the species most often associates with to build an accurate and biologically realistic picture of its ranging patterns and habitat use. Conservation managers and regulatory agencies can, in turn, use this scientific information to gauge and improve the

effectiveness of existing and proposed measures to protect important animal populations, such as habitat conservation zoning, reserve boundaries and wildlife corridors.

Biologists typically define non-random and restricted patterns of space use of individual animals using a home range estimator. A wide range of empirical home range estimators have been developed to model animal spatial behaviors, and more recently, the interplay between the environment and an animal's cognitive map of its habitat [5, 6]. Home range estimators typically summarize patterns of animal spatial behaviors as densities of space use relative to time derived from nonparametric estimates of animal locations across a landscape. A prime example is the spatial probability density function known as the utilization distribution (UD), that estimates the probabilities of where an animal might have occurred at an arbitrary time during the period the animal was observed [7, 8]. The kernel density estimator (KDE), which uses a weighted sum of kernels placed over observed animal locations [9], has become a standard technique for estimating home ranges. However, KDEs are criticized for sometimes excluding areas that have been used by animals (type I errors) or including areas that have not been used (type II errors) and biotelemetry data sets with relatively short time intervals between locations need to be subsampled to ensure independence [10].

The Brownian bridge approach [11] provides an alternative KDE that integrates kernels over time along a movement path interpolated between observed locations. Benhamou [12] distinguished Worton's location-based kernel density estimators (LKDE) from movement-based kernel density estimators (MKDE), which includes Brownian bridge and biased random walk models [11, 12]. Unlike LKDEs, MKDEs account for time between consecutive observations, do not requiring independent samples from the UD, and more realistically represent the space used by an animal.

The coevolution of biologgers and home range estimators is bringing inferences on animal space use closer to biological reality. However, current estimators fail to capitalize on the 3D profiles offered by modern GPS biotelemetry datasets. Animal space-use is multi-dimensional and can be characterized within two *x* and *y* planar spatial dimensions, as well as a *z*-dimension representing altitude (for flying or arboreal species), elevation (for terrestrial species), or depth (for aquatic species). Biologgers have been providing multidimensional data on animal movements for decades. Yet surprisingly, there are almost no modeling techniques that explicitly integrate the 3rd spatial dimension of the *z*-coordinate into quantitative characterizations of animal spatial behaviors. Disregarding the z dimension greatly limits our understanding of the vertical component of animal ranging patterns and restricts our ability to define and predict how animals move through landscapes and select and use habitats [13]. Traditional 2D home range estimators may also misrepresent the space use of animals that occupy habitats with a strong vertical component.

While 3D visualization and modeling tools have been used extensively in disciplines like chemistry, climatology, and physics, biologists are only beginning to recognize the theoretical and applied value of incorporating the vertical aspect into analyses of animal space use. We recently advanced the movement-based kernel density estimator method of calculating animal home ranges by extending it into three-dimensions (3D MKDE [14]. 3D models of animal space-use can enhance conservation strategies for mitigating the effects of anthropogenic impacts on vagile wildlife populations. For example, analyzing the 3D MKDE home range volumes of birds and bats in relation to the spatiotemporal distribution of aircraft flight paths, buildings, or wind turbines will provide more accurate estimates of collision risk than 2D models. Improved understanding of the 3D spatial behaviors of the many aquatic animals currently being tracked with biologgers, such as marine turtles, would help managers to minimize their incidental capture by fisheries. 3D MKDEs could also be incorporated into predictive models of wildlife exposure to soil, air and water borne contaminants.

2. MKDE ALGORITHM

In this section we provide the formalism for the 3D MKDE method. Our objective is to use observed location data to develop a density estimate describing space-use of an animal at the times it was not observed. We follow the method of Tracey, Sheppard, et al., who describe the development method in great detail and provide several illustrative examples [14]. Here, we describe computational aspects of the method in greater depth.

2.1 Data set

Suppose we have made *n* observations of the locations of an individual animal. We index observations by *m=0, ..., n-1*. The m^{th} observation consists of an *x*-coordinate, a *y*-coordinate, a *z*-coordinate, and a time which we denote x_m, y_m, z_m, t_m, respectively. We require the observations to be ordered in time such that $t_m\text{-}1 < t_m < t_{m+1}$ for any *m*. Further, we assume that the observed locations subject to observation error described by normal distributions with means x_m, y_m, z_m, and variances δ_m^2, δ_m^2, and ε_m^2, respectively. Notice that we assume that the error variances are the same for the observed values of x and y. The observation error variances δ_m^2 and ε_m^2 must be estimated independently, and are typically either provided by the manufacturers of the telemetry equipment or estimated from field trials. In some cases, they are the same for all m when observation-specific variances cannot be estimated. The m^{th} move step is defined by two consecutive observations m and *m+1*. In order to ensure that we do not include move steps with unusually long time intervals, for example when the GPS receiver fails to acquire one or more position fixes, we only use move steps where $\Delta t_m = t_{m+1} - t_m \le \Delta t_{\max}$ given a user-specified maximum allowed time of $\Delta t_{\max}$. We use *I(m)* = 1 if this condition is met, and 0 otherwise, as an indicator function.

2.2 Spatial Domain

We compute the probability for every voxel (3D cell) on a regular grid in three-dimensional space. We index rows by *i=0,…,I-1*, columns by *j=0,…,J-1*, and levels by *k=0,…,K-1* where *I*, *J*, and *K* are the number of rows, columns, and levels, respectively. Rows, columns, and levels correspond to the *y*-, *x*-, and *z*-dimensions of the 3D MKDE, respectively. We index voxels by $v=i+j\times I+k\times I\times J$. V denotes the random variable for the voxel in which the individual is found. We define the 3D regular grid on which the density will be computed by 1D arrays of voxel center coordinates in the *x*, *y*, and *z* spatial dimensions denoted *xGrd*, *yGrd*, and *zGrd*. Because we assume a regular grid, the half the lengths of the sides of a voxel in each dimension are h_x = *(xGrd[1]-xGrd[0])/2*, h_y = *(yGrd[1]-yGrd[0])/2*, and h_z = *(zGrd[1]-zGrd[0])/2*. We can determine

the extent of the grid by [*xGrd[0]- h_x, xGrd[J-1]+ h_x*] for the *x*-dimension and similarly for the other spatial dimensions.

Often, we must bound the physical space an animal can occupy in the *z*-dimension in order to produce a more realistic 3D MKDE [14].We can set a lower and upper bound the density in the z-dimension at an location (*x,y*) by *a*(*x,y*) and *b*(*x,y*), respectively. These bounds can be described by two 2D rasters, but whether or not the MKDE must be bounded above or below is case specific. For example, we use a digital elevation model to set the lower boundary for a condor.

2.3 Algorithm

We estimate the 3D MKDE using a trivariate normal kernel integrated over time for each observed move step. The kernel describing the probability density at time *t* and location (*x,y,z*) is

$$f_{XYZ}(x,y,z \mid \mathbf{u}(t),\Sigma(t)),$$

where $\mu(t)$ is a vector of means and $\Sigma(t)$ is the 3 × 3 covariance matrix. We assume that the joint density can be represented by the product of the univariate densities for each spatial dimension as

$$f_{XYZ}(x,y,z \mid \mathbf{u}(t),\Sigma(t)) = f_X(\mathrm{x} \mid \mu_1(t),\sigma^2_{xy}(t))\times f_Y(\mathrm{y} \mid \mu_2(t),\sigma^2_{xy}(t))\times f_Z(\mathrm{z} \mid \mu_3(t),\sigma^2_z(t))$$

That is, we assume independence in each spatial dimension, conditional on the constraints in the z-dimension. This assumption can be exploited to increase the computational efficiency of the 2D or 3D MKDE calculations.

To obtain the average probability that the animal was in a given voxel *v* at some arbitrary time during the time interval [t_m, t_{m+1}), given that the voxel *v* is within the allowed range in the z-dimension, we calculate

$$P_m(\mathrm{v}=V) = \frac{1}{W_m}\int_{t_m}^{t_{m+1}}\int_{x_j-h_x}^{x_j+h_x}\int_{y_i-h_y}^{y_i+h_y}\int_{z_k-h_z}^{z_k+h_z} f_{XYZ}(\mathrm{x},\mathrm{y},\mathrm{z} \mid \mathbf{u}(t),\Sigma(t))dzdydxdt.$$

However, if the voxel v is not in the allowed range in the z-dimension, then $P_m(v=V)=0$.

Next, we sum over all valid observed move steps by

$$P(v=V) = \frac{1}{W}\sum_{m=0}^{n-1} W_m P_m(v=V) I(m).$$

Finally, we obtain the normalization constant by calculating

$$W_m = \int_{t_m}^{t_{m+1}}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{a(x,y)}^{b(x,y)} f_{XYZ}(x,y,z \mid \mathbf{u}(t),\Sigma(t))dzdydxdt$$

and

$$W = \sum_{m=0}^{n-1} W_m I(m).$$

In practice, this amounts to dividing the non-normalized probability for each voxel by the sum of the non-normalized probabilities over all voxels.

In addition, two variance parameters, η^2 and γ^2, must be estimated form the data as described by Tracey, Sheppard, et al. [14]. Finally, the user must specify the number of integration time steps (*nSteps*) per move step.

Computationally, we iterate through each move step m, numerically integrate over time using the trapezoid rule by iterating through *nStep* integration time steps per move step, then iterate through the x, y, and z dimensions of the regular grid. This process, together with the auxiliary functions needed to calculate means, variances and integrals, are described by the function *Compute3DMKDE()* in the Appendix.

The function KernelMean() describes the algorithm for calculating the vector of kernel means as a function of time. This function calculates the expected location of the animal at a time t on the interval [t_m, t_{m+1}) for move step *m*. The function KernelVar() describes the algorithm for computing the diagonal elements of the covariance matrix

$$\Sigma(t) = \begin{bmatrix} \sigma^2_{xy}(t) & 0 & 0 \\ 0 & \sigma^2_{xy}(t) & 0 \\ 0 & 0 & \sigma^2_z(t) \end{bmatrix},$$

where $\sigma^2_{xy}(t)$ is the time-dependent kernel variance for the *x*-dimension and *y*-dimension $\sigma^2_z(t)$ is the time-dependent variance for the *z*-dimension. These variances incorporate observation error and uncertainty in the animal's position at unobserved times *t* between t_m and t_{m+1}. Finally, function IntegrateNormal() provides the method for integrating the univariate normal density for each spatial dimension. The function Phi() takes a standardized normal random variable as its argument and returns the corresponding value of the standard normal cumulative density function (c.d.f.)

3. OPTIMIZATION AND PARALLELIZATION OF MKDE ALGORITHM

One downside of implementing a fully 3D MKDE is the greatly increased computational expense relative to simpler, lower-dimensional methods. We decided to take a two-pronged approach to making the computations more tractable: restructuring of the code to optimize the underlying algorithm and shared memory parallelization of the key loops.

3.1 Program restructuring

The computationally demanding portion of the original version of the code logically consisted of four sets of nested loops

1. Groups: sets of observations for a particular animal over a given time
2. Voxels: 3D discretization of the animal domain
3. Observations: GPS measurements at particular time
4. Interpolations: Interpolated values of location at times between observations

At the innermost loop, a contribution is made to the voxel density from a kernel function that depends on both the horizontal and vertical distances between the voxel and the interpolated point. Although the original code makes optimal use of cache by accessing the voxels in their storage order, it results in the repeated calculation of kernel parameters that do not depend on the distance to the voxel. For example, the two variances that are needed to evaluate the kernel depend only on

the distance between the interpolated point and the bounding observations. Reordering so that the loop over voxels occurs at the innermost level of nesting makes it easier to avoid these repeated calculations

Variances, coordinates of the interpolated point and other kernel parameters are now evaluated once and then used for all of the voxels. After reordering the loops, we rewrote the loop over voxels as an equivalent set of nested loops over the three spatial dimensions. This makes additional opportunities for the avoidance of unnecessary calculations more obvious. For example, quantities that do not depend on the *z*-coordinate, such as the squared distance in the *xy*-plane between voxel and kernel origin can be pre-computed before entering the new innermost loop and repeatedly used. These speedups make it practical to implement more meaningful, but also more computationally expensive, quantities such as the integral over the voxel volume to yield voxel probabilities.

3.2 Cutoff on kernel evaluation

As originally written, the kernel was evaluated at every voxel. The contribution from a kernel falls off rapidly with the distance from the kernel origin, particularly for interpolated points that are close to the observations where the variance is low. The restructuring of the code described in the previous section makes it much more straightforward to apply a variance dependent cutoff distance. Rather than applying the cutoff within the innermost loop, bounds on the voxel loop indices are pre-determined thereby limiting iterations to only those voxels that will be within the cutoff.

The application of a cutoff changes the inherent scalability of the code with problem size. Given n observations and N voxels, the original run time scaled as $O(nN)$. The use of a cutoff results in a fixed amount of work per observation and reduces the scaling to $O(n)$. For a simple uniform cutoff of d cells in all three spatial dimensions and ignoring boundary effects, the time to solution will be shortened by a factor of $N/(8d^3)$. The scaling behavior when using a more realistic variance-dependent cutoff will be more complicated and depends on the characteristics of the data set. The relative error when applying a cutoff will be problem dependent, but for our test problem using a tight cutoff of 20 voxels in each direction resulted in only 0.08% error relative to original code. At a 50-voxel cutoff, no error was detected.

3.3 Shared memory parallelization

While an MPI version of the code was already available, we felt that the shared memory version might be preferable for a number of reasons. First, the natural chunk of work is at the level of an individual animal observed over a period of time (e.g. condor-month or manatee-week). Chunks could be distributed to different processes, each of which runs as a multi-threaded job on a single compute node. Although the scalability of the shared memory version is limited by the number of compute cores available in a node, this should not be a handicap assuming that the time to solution for an individual chunk is sufficiently short and that the researchers are more concerned with throughput rather than minimizing the time to solution for a single unit of work.

Second, given the nature of the computations, where each voxel can be updated independently, a threaded version of the code can be very simple to develop, maintain and deploy. In fact, after restructuring the code to achieve better performance as described in the previous section, the parallelization involved the addition of just a single OpenMP directive before the outermost (*x*-coordinate) of the three spatial dimension loops. Our application is targeted at a wide range of users with varying degrees of sophistication and access to hardware ranging from laptops to supercomputers. By avoiding the need to locally install an MPI library, the user only needs to download a single binary. For those users who decide to build their own executables from source, they would only need access to an OpenMP capable C++ compiler.

Finally, a shared memory implementation may potentially be more efficient, particularly if the parallel portion of the code contains no serial or critical regions. Even within a node, the MPI overhead can degrade performance unless a high-performance intra-node communications library such as LiMIC [15] is available.

3.4 Benchmarks

The MKDE algorithm is in a state of active development and code modifications made as part of this collaboration are reintegrated into newer versions of the software. As a consequence, it is not practical to reverse engineer later versions to match the original structure. The following benchmarks should be considered as representative snapshots of an earlier version of the code. In addition, the optimal way to apply a variance-dependent cutoff is still being investigated. For simplicity, we restrict our study to a fixed-size cutoff. Nonetheless, these timings provide a general overview of the impact of our changes on the performance.

Table 1. Benchmark of test problem based on 337 Condor observations from September 2010. Timings for both original and optimized codes obtained from parallel runs using 16 cores on a single Gordon compute node. All executables built with Intel C++ compiler (icpc) version 13.0.1 with -O3, -xHOST and -mkl. Optimized version of code includes reordering of loops, pre-calculation of reused quantities and other statement-level modifications. In rows 4-9, d=D refers to the size of the fixed cutoff.

version	t(s)	speedup
Original	1829	1.0
Optimized	139	13.2
Opt w/ d=100	25.3	72.3
Opt w/ d=60	9.5	192.5
Opt w/ d=50	7.1	257.6
Opt w/ d=40	5.2	351.7
Opt w/ d=30	2.7	677.4
Opt w/ d=20	1.4	1306.4

A subsequent round of optimizations took advantage of the fact that the kernel can be separated into a product of functions that depend separately on the horizontal (x and y) and vertical (z) distances from the kernel origin. A vector of the z-dependent function results can be calculated once before entering the loop over voxels and reused at every location in the xy-plane. These timings are presented separately in Table 2 since they rely on a regular grid, the usual case, and would not be applicable to an implementation of the algorithm on an irregular grid.

4. VISUALIZATION

The output from the code is a cell-centered scalar probability density on a 3D rectilinear uniform grid. This was written out by the original code in VTK ASCII format and visualized in Paraview. The original process was refined as follows making it possible to visualize results in both Paraview and VisIt software.

4.1 Restructuring output

The code was restructured to write output as a brick of doubles in binary format, wrapped with an XDMF metadata header to

Table 2. Benchmark of test problem based on 337 Condor observations from September 2010, with additional optimizations that take advantage of kernel function being separable into product of z-dependent and xy-dependent terms.

version	t(s)	speedup
Original	1829	1.0
Optimized	42.5	43.0
Opt w/ d=100	3.87	472
Opt w/ d=60	1.67	1095
Opt w/ d=50	1.25	1463
Opt w/ d=40	0.87	2102
Opt w/ d=30	0.67	2729
Opt w/ d=20	0.46	3976

describe the binary data as a cell centered scalar 3D unigrid. The reduction in file size relative to the original ASCII data was found to be 85%, a ratio that held for a range of original file sizes. Another output restructuring was made to compute voxel probabilities by integrating the kernels over the voxel rather than voxel-centered probability densities. With earlier code, the voxel-centered probability densities were extremely small and close to hardware precision epsilon. This caused reliability concerns, especially when using the output data in other tools.

4.2 Isosurface Extraction

One key interest for the scientist is to visualize and explore the bird's habitat envelope at multiple probability densities. This required computation of corresponding isovalues, which were calculated as follows

- Create 1D indexed array A, containing voxel probabilities
- Create 2D array B, where B[1] contains probabilities and B[2] contains corresponding index from array A
- Sort 2D array B in place with probabilities as key in increasing order
- Create another 2D array C with C[1] containing cumulative densities from sorted array B[1] and C[2] containing corresponding index in sorted array B[2]
- Pick a percentage of the total density of interest, for example 95%, then perform a lookup in C[1] for 95% to identify original index in C[2].
- Look up the scalar probability density in array A at index found above. This is the isovalue for 95% envelope
- Create an isosurface at above isovalue

Figure 1 shows the trajectory and isosurfaces with probability envelopes at 99%, 95%, 90%, 75% and 50%.

4.3 Adding Context

While the visualization of scalar probability density is useful, it requires context to provide further insight about a bird's location and habitation zone. To accomplish this a digital elevation model (DEM) of the terrain was added and further refined by addition of texture map marking features like shorelines, etc. Other elements like wind turbines are also being added. This has enabled the scientist to explore the habitat zone in geographic context.

4.4 Visualization Discussion

Visualization was used to compare the output from original and revised MKDE code, as well as with the observed trajectories.

We are able to observe the possible nest locations of the birds and their habitat in 3D. Currently, the visualization is setup as a pipeline in VisIt software, which reads data including temporal probability density, and corresponding isovalues and DEM. We are training the scientist to perform these visualizations on local desktop as well as remotely on Gordon supercomputer at SDSC.

5. FUTURE WORK

The large improvements in the code performance make it possible to carry out calculations that were previously not feasible. Obviously it will be easier to process larger amounts of data, obtained either by tracking greater numbers of animals or recording observations at a higher frequency. Enhancement to the MKDE implementation and their impact on the final results can also be evaluated more quickly. One novel idea that was recently proposed was to calculate probability contour volumes using a sliding time window. This might entail an order or more magnitude of work, but would eliminate or reduce the sharp discontinuities observed in the visualization when jumping from one month to the next. We would also like to pipeline the visualization such that few predetermined plots are created after each computation. We also plan to leverage SeedMe.org infrastructure [16] to share the new results ubiquitously. Finally, a very fast algorithm allows calculations to be launched and have results returned in just seconds. This might be an extremely valuable capability for a conservation biologist in the field who wants immediate information on the recent range of one or more animals. Finally, researchers are beginning to consider calculations involving multiple animals and overlaps in the space use. These suggestions are just a beginning and we are hoping that the end users will drive new use cases based on the enhanced capabilities.

6. ACKNOWLEDGMENTS

We acknowledge the following California condor funding agencies and collaborators: United States Fish and Wildlife Service, Instituto Nacional de Ecologia, Comision Nacional Para El Conocimiento y Uso de la Biodiversidad, Secretaria de Medio Ambiente y Recursos Naturales, Wildcoast/Costasalvaje, Sempra Energy, Michael Wallace, Lisa Nordstrom and the SDZG condor field team. We thank Jesse Lewis for his helpful discussions related to Brownian bridge models. RSS, MT, AC and GL also received partial support from NSF grant: OCI #0910847 Gordon: A Data Intensive Supercomputer. Any use of trade, product, or firm names is for descriptive purposes only and does not imply an endorsement by the U.S. Government.

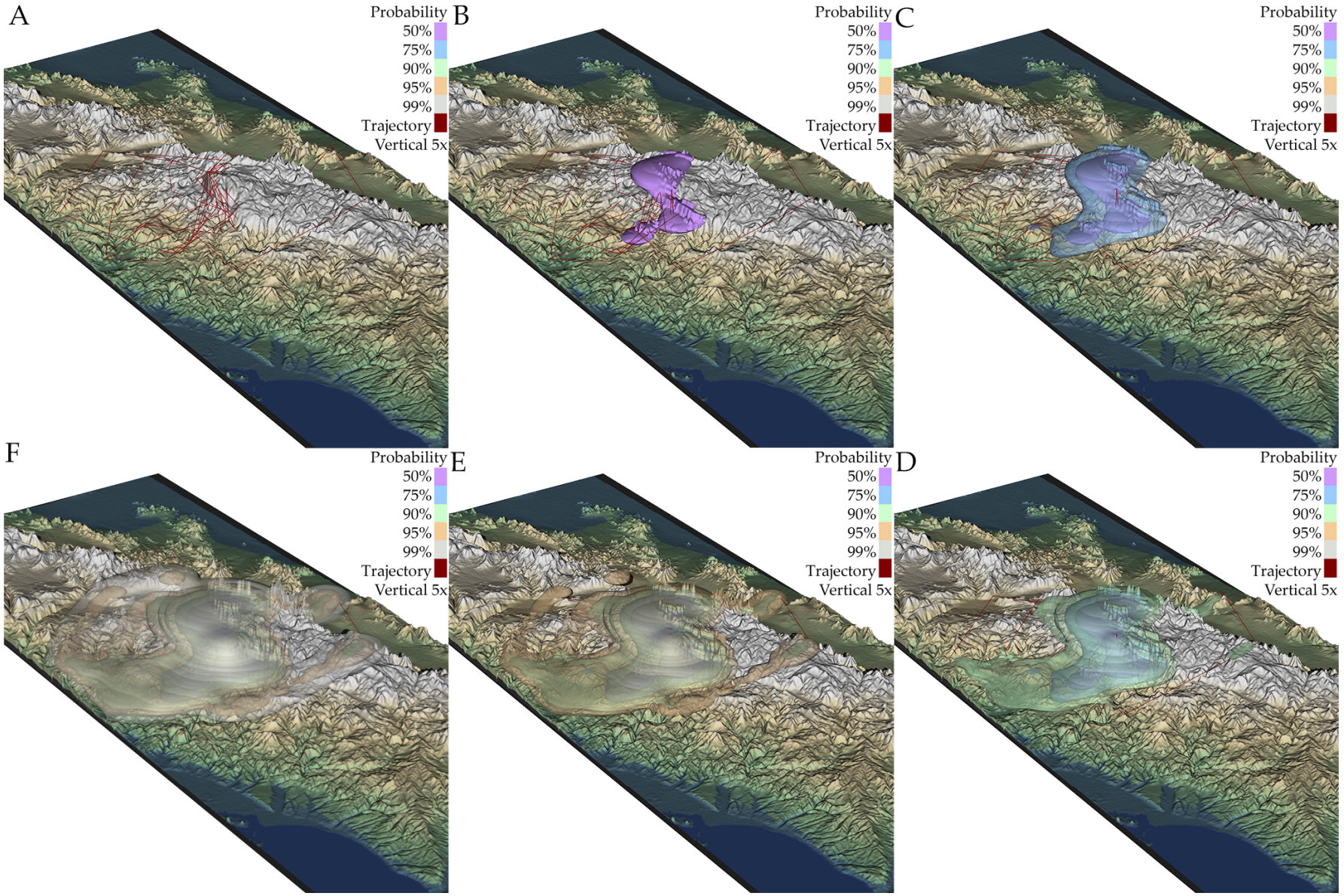

Figure 1 The image depicts a composite visualization for a volume 98.5k x 223.5 km x 5.5km, where vertical axis has been scaled by 5x. Panel (A) shows trajectory of a condor in March, 2010. Panel (B) is same as (A) with addition of isosurface encompassing 50% probability region. Panels (C-F) successively add isosurfaces at 75%, 90%, 95% and 99% probabilities.

7. REFERENCES

[1] Cagnacci, F., Boitani, L., Powell, R. A. and Boyce, M. S. Animal ecology meets GPS-based radiotelemetry: a perfect storm of opportunities and challenges. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 365, 1550 (Jul 27 2010), 2157-2162.

[2] Hebblewhite, M. and Haydon, D. T. Distinguishing technology from biology: a critical review of the use of GPS telemetry data in ecology. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 365, 1550 (Jul 27 2010), 2303-2312.

[3] Tomkiewicz, S. M., Fuller, M. R., Kie, J. G. and Bates, K. K. Global positioning system and associated technologies in animal behaviour and ecological research. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 365, 1550 (Jul 27 2010), 2163-2176.

[4] Sheppard, J. K., Walenski, M., Wallace, M. P., Velazco, J. J. V., Porras, C. and Swaisgood, R. R. Hierarchical dominance structure in reintroduced California condors: correlates, consequences, and dynamics. *Behavioral Ecology and Sociobiology* 67 (2013), 1227-1238.

[5] Powell, R. A. and Mitchell, M. S. What is a home range? *Journal of Mammalogy* 93 (2012), 948-958.

[6] Spencer, W. D. Home ranges and the value of spatial information. *Journal of Mammalogy*, 93 (2012), 929-947.

[7] Silverman, B. W. *Density estimation for statistics and data analysis*. Chapman and Hall, London ; New York, 1986.

[8] Van Winkle, W. Comparison of several probabilistic home range models. *Journal of Wildlife Management* 39 (1975), 118-123.

[9] Worton, B. J. Kernel methods for estimating the utilization distribution in home-range studies. *Ecology*, 70 (1989), 164-168.

[10] Fieberg, J. and Börger, L. L. Could you please phrase "home range" as a question? *Journal of Mammalogy* 93 (2012), 890-902.

[11] Horne, J. S., Garton, E. O., Krone, S. M. and Lewis, J. S. Analyzing animal movements using Brownian bridges. *Ecology*, 88, 9 (Sep 2007), 2354-2363.

[12] Benhamou, S. Dynamic approach to space and habitat use based on biased random bridges. *PloS ONE*, 6, 1 (2011), e14592.

[13] Belant, J. L., Millspaugh, J. J., Martin, J. A. and Gitzen, R. A. Multi-dimensional space use: the final frontier. *Frontiers in Ecology and the Environment*, 10, 1 (2012), 11-12.

[14] Tracey, J. A., Sheppard, J. K., Zhu, J., Wei, F., Swaisgood, R. R. and Fisher, R. N. *Movement-based Estimation and Visualization of Space Use in 3D for Wildlife Ecology and Conservation*., PLoS ONE (in revision).

[15] Jin, H.-W., Sur, S., Chai, L. and Panda, D. K. *Limic: Support for high-performance mpi intra-node communication on linux cluster*. In *IEEE International Conference on Parallel Processing ICPP 2005*, (2005).

[16] Chourasia, A., Wong-Barnum, M. and Norman, M. L. *SeedMe preview: your results from disk to device.* In *Proceedings of XSEDE13 Conference* (Atlanta, GA 2013).

8. APPENDIX : CODE OUTLINE

In the pseudocode, the arguments x, y, z, and t are 1D arrays of double values representing our observed data. That is, the m^{th} observation (denoted x_m, y_m. z_m, and t_m in the text) is (x[m], y[m], z[m], t[m]). The arguments obsVarXY and obsVarZ (denoted δ_m^2 and ε_m^2 in the text) are 1D arrays of double variables where the m^{th} element is the observation error variance for the move step in the (x,y)-dimensions and z-dimension, respectively. The arguments mvVarXY and mvVarZ are single double precision variances for the move variances parameters (η^2 and γ^2 in the text) in the (x,y)-dimensions and z-dimension, respectively. The 1D arrays of double precision floating point variables xGrd, yGrd, and zGrd define the 3D regular grid on which we compute the 3D MDKE, as described in the text. From the 2D arrays lower[][] and upper[][] which correspond to $a(x,y)$ and $b(x,y)$ in the text, we create the 1D integer arrays lowerVox[] and upperVox[], which contain indexes into zGrid for each i,j, where lower[i][j] corresponds to lowerVox[i+j×I] and upper[i][j] corresponds to lowerVox[i+j×I]. The value of lowerVox[i+j×I] equals the number of elements in zGrd less than lower[i][j], while upperVox[i+j×I] equals the number of elements in zGrd less than upper[i][j] minus one. The double-precision argument tMx ($\Delta t_{\max}$ in the text) is the maximum allowed time duration of a move step that can be included in the MKDE estimation. Finally, nStep is the number of numerical integration time steps per move step. In the functions KenrelMean() and KernelVar(), t is the time in the numerical integration, x0, y0, z0, and t0 are the observed data at the beginning of the move step while x1, y1, z1, and t1 are the observed data at the end of the move step (i.e. observations m and $m+1$ for the m^{th} move step).

```
Compute3DMKDE(x: double[], y: double[],
              z: double[], t: double[],
              tMx: double, obVarXY: double[],
              obVarZ: double[], mvVarXY: double,
              mvVarZ: double, xGrd: double[],
              yGrd: double[], zGrd: double[],
              lowerVox: int[], upperVox: int[],
              tMx: double, nStep: int)
              : double[]

nObs ← length(t)

(nx, ny, nz) ← length(xGrd, yGrd, zGrd)

// we assume a regular grid
xsz ← xGrd[1] – xGrd[0]
ysz ← yGrd[1] – yGrd[0]
zsz ← zGrd[1] – zGrd[0]

nv ← nx*ny*nz
mkde ← double[nv] // init all elements to 0
W ← 0

for (m in 0 to nObs - 2)
  deltaT ← t[m+1] – t[m]
  if (deltaT <= tMx)
    tStep ← deltaT/nStep
    tmpIntegral ← 0
    for (s in 0 to nStep)
      tCurrent ← t[m] + s*tStep
      mu ← KernelMean(tCurrent, t[m], t[m+1],
                      x[m], x[m+1],
                      y[m], y[m+1],
                      z[m], z[m+1])
      var ← KernelVar(tCurrent, t[m], t[m+1],
                      obVarXY[m], obVarXY[m+1],
                      obVarZ[m], obVarZ[m+1],
                      mvVarXY, mvVarZ)

      for (i in 0 to nx – 1) //Parallel loop
        probX ← IntegrateNormal(xGrd[i]-xsz,
                  xGrd[i] + xsz, mu[0], var[0])
        for (j in 0 to ny – 1)
          probY ← IntegrateNormal(yGrd[j]-ysz,
                    yGrd[j] + ysz, mu[1], var[1])
          kLo ← lowerVox[i + j*nx]
          kHi ← upperVox[i + j*nx]
          for (k in kLo to kHi)
            probZ ← IntegrateNormal(zGrd[k]-zsz,
                      zGrd[k]+zsz, mu[2], var[2])
            v ← i + j*nx + k*nx*ny
            probXYZ ← probX*probX*probZ
            if (s == 0 or s == nStep)
              tmp ← tStep*probXYZ/2
            else
              tmp ← tStep*probXYZ
            end if
            mkde[v] ← mkde[v] + tmp
            W ← W + tmp
          end for // z
        end for // y
      end for // x

    end for // int step
  end if // cond
end for // move

mkde ← mkde/W // normalize probability
return mkde
```

```
KernelMean(t: double, t0: double, t1: double,
           x0: double, x1: double,
           y0: double ,y1: double,
           z0: double, z1: double)
           : double[3]

mu ← double[3]
deltaT ← t1 – t0
alpha ← (t – t0)/deltaT
mu[0] ← x0 + (x1 – x0)*alpha
mu[1] ← y0 + (y1 – y0)*alpha
mu[2] ← z0 + (z1 – z0)*alpha
return mu

KernelVar(t: double, t0: double, t1: double,
          obVarXY0: double, obVarXY1: double,
          obVarZ0: double, obVarZ1: double,
          mvVarXY: double, mvVarZ: double)
          : double[3]

var ← double[3]
deltaT ← t1 – t0
alpha ← (t – t0)/deltaT
xyVar ← deltaT*alpha*(1–alpha)*mvVarXY +
        (1–alpha)*(1–alpha)*obVarXY0  +
        alpha*alpha*obVarXY1
var[0] ← xyVar
var[1] ← xyVar
var[2] ← deltaT*alpha*(1–alpha)*mvVarZ +
          (1–alpha)*(1–alpha)* obVarZ0 +
          alpha*alpha* obVarZ1
return var

IntegrateNormal(x0: double, x1: double,
                mu: double, var: double)
                : double
prob ← Phi((x1-mu  / sqrt(var)) –
       Phi((x0- mu)/ sqrt(var))
return prob
```